HPIM: <u>H</u>eterogeneous <u>P</u>rocessing-<u>I</u>n-<u>M</u>emory-based Accelerator for Large Language Models Inference

Cenlin Duan, Jianlei Yang, *Senior Member, IEEE,* Rubing Yang, Yikun Wang, Yiou Wang, Lingkun Long, Yingjie Qi, Xiaolin He, Ao Zhou, Xueyan Wang, *Member, IEEE,* and Weisheng Zhao, *Fellow, IEEE*

Abstract—The deployment of large language models (LLMs) presents significant challenges due to their enormous memory footprints, low arithmetic intensity, and stringent latency requirements, particularly during the autoregressive decoding stage. Traditional compute-centric accelerators, such as GPUs, suffer from severe resource underutilization and memory bandwidth bottlenecks in these memory-bound workloads. To overcome these fundamental limitations, we propose HPIM, the first memory-centric heterogeneous Processing-In-Memory (PIM) accelerator that integrates SRAM-PIM and HBM-PIM subsystems designed specifically for LLM inference. HPIM employs a software-hardware co-design approach that combines a specialized compiler framework with a heterogeneous hardware architecture. It intelligently partitions workloads based on their characteristics: latency-critical attention operations are mapped to the SRAM-PIM subsystem to exploit its ultra-low latency and high computational flexibility, while weight-intensive GEMV computations are assigned to the HBM-PIM subsystem to leverage its high internal bandwidth and large storage capacity. Furthermore, HPIM introduces a tightly coupled pipeline strategy across SRAM-PIM and HBM-PIM subsystems to maximize intra-token parallelism, thereby significantly mitigating serial dependency of the autoregressive decoding stage. Comprehensive evaluations using a cycle-accurate simulator demonstrate that HPIM significantly outperforms state-of-the-art accelerators, achieving a peak speedup of up to $34.3\times$ compared to the NVIDIA A100 GPU. Moreover, HPIM exhibits superior performance over contemporary PIM-based accelerators, highlighting its potential as a highly practical and scalable solution for accelerating large-scale LLM inference.

Index Terms—Large Language Models, Heterogeneous Processing-In-Memory, HBM-PIM, SRAM-PIM

I. INTRODUCTION

Large Language Models (LLMs), such as GPT family [1, 2] and LLAMA family [3–5], have dramatically transformed the field of artificial intelligence, enabling unprecedented capabilities in conversational agents [6, 7], text generation [8], and code synthesis [9–11]. However, the deployment of these
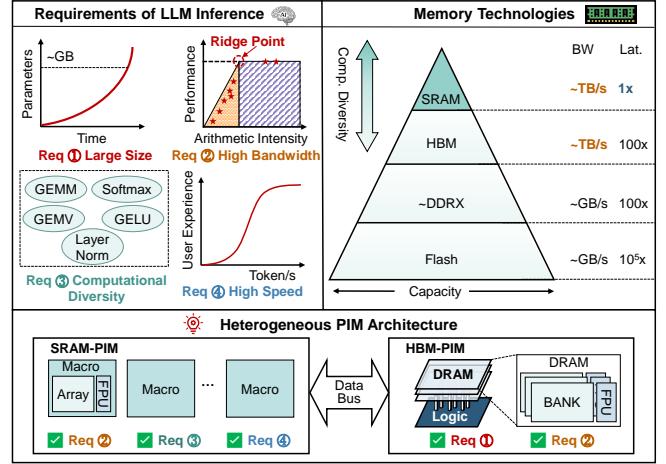
Fig. 1. The diverse requirements of LLM inference, the corresponding trade-offs in "PIM pyramid", and the advantages of heterogeneous PIM architecture.

models is fundamentally limited by their inherently low computational intensity, particularly during the autoregressive decoding phase. This memory-bound characteristic imposes a severe bandwidth bottleneck on conventional accelerators, such as GPUs, which are optimized for compute-intensive workloads. Such a fundamental mismatch directly leads to severe compute underutilization, degrades overall throughput and efficiency, and ultimately motivates a paradigm shift towards novel architectural solutions.

To address these challenges, Processing-In-Memory (PIM) [12–17] has emerged as a promising architectural paradigm. PIM aims to mitigate the data movement bottleneck by integrating computational logic directly within or near memory arrays to exploit massive internal memory bandwidth. It can be implemented across a spectrum of memory technologies, each presenting a distinct trade-off between capacity, bandwidth, computational diversity, and latency, as illustrated by the "PIM pyramid" in Fig. 1. At the top of the pyramid, SRAM-PIM [18, 19] offers ultra-low latency ($\sim 1\times$), high internal bandwidth, and its logic-process compatibility makes it particularly suitable for integrating complex functions. However, due to limited storage density ($\sim$MB), SRAM-PIM alone is insufficient for meeting the massive memory requirements of LLM inference. In contrast, HBM-PIM [20, 21] provides a balance of high bandwidth and capacity. Yet, it is hampered by higher latency and prohibitive logic integration costs. This fundamental integration challenge is also inherent to capacity-oriented technologies like LPDDR [22, 23] and Flash [24, 25].
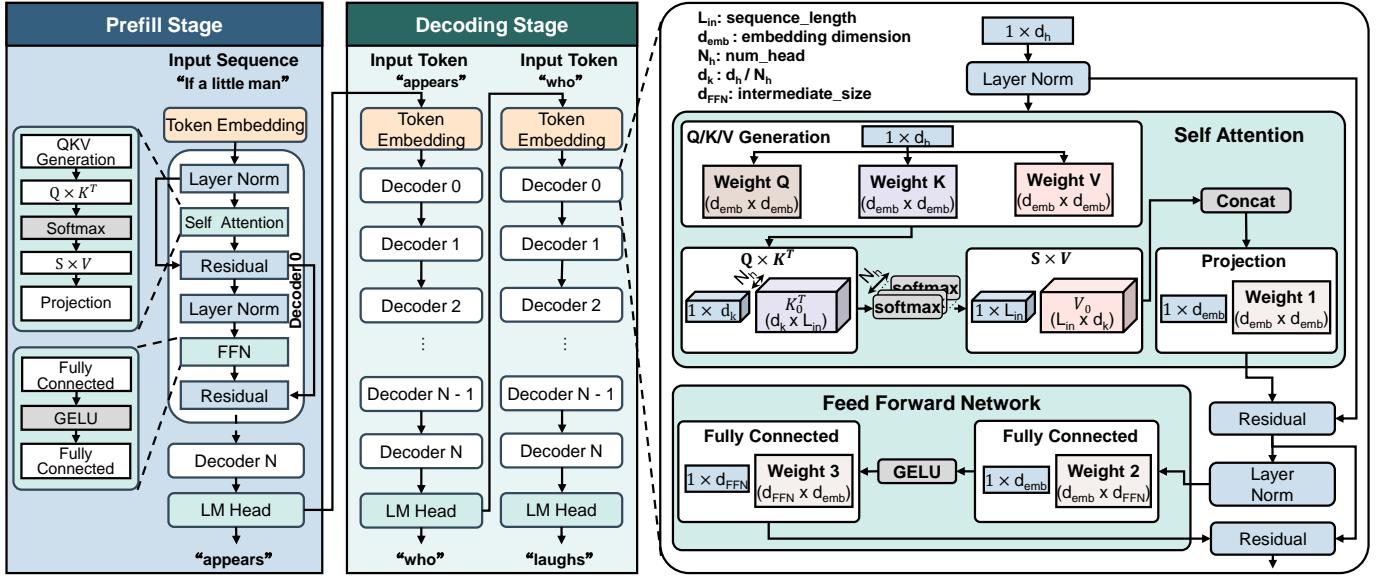
Fig. 2. Model architecture and inference process of LLM.

Meanwhile, they are further hampered by substantially lower bandwidth and greater latency, rendering them unsuitable for high-performance LLM acceleration.

The above analysis reveals that a single memory technology can not simultaneously satisfy the diverse requirements of LLM inference, including large model size, high bandwidth, computational diversity, and low latency. This directly motivates the exploration of a heterogeneous PIM architecture to accelerate LLM inference by synergistically combining different memory technologies. Based on our analysis, a combination of SRAM-PIM and HBM-PIM emerges as the most promising approach to meet these diverse demands. Therefore, we propose HPIM, a heterogeneous PIM architecture, as depicted at the bottom of Fig. 1. This design leverages the complementary strengths of both memory types through a clear partitioning strategy. This balanced approach offers a practical and scalable path forward for accelerating the LLM inference. Our contributions are as follows:

- **Memory-Centric Heterogeneous PIM Architecture:** We propose HPIM, the first memory-centric heterogeneous PIM architecture to integrate a low-latency, computationally-flexible SRAM-PIM and a high-bandwidth, high-capacity HBM-PIM for single-batch LLM inference. This novel architecture leverages parallel execution between these two specialized subsystems to overcome the memory bandwidth bottlenecks and serial dependencies inherent in autoregressive decoding.
- **Adaptive Workload Partitioning and Intra-Token Parallelism:** We introduce a novel hardware-aware workload partitioning and scheduling strategy that allocates compute-intensive, latency-sensitive tasks to SRAM-PIM, and weight-intensive operations to HBM-PIM. This strategy significantly minimizes inter-subsystem data movement and communication overhead. Furthermore, to overcome token-level serialization, HPIM employs a tightly-

coupled execution pipeline across subsystems that exploits fine-grained intra-token parallelism. This mechanism enables overlapping of QKV generation, attention, and FFN operations, significantly shortening the decoding critical path.
- **Comprehensive Evaluation:** We evaluate HPIM on a cycle-accurate simulator across model scales from the OPT family (350M∼30B). Results show that HPIM achieves a peak speedup of up to $22.8\times$ compared to an NVIDIA A100 GPU. Furthermore, HPIM delivers $1.50\times$ lower latency than IANUS and up to $5.76\times$ higher throughput than CXL-PNM, demonstrating its superiority over state-of-the-art (SOTA) PIM accelerators.

The remainder of this paper is organized as follows. Section II reviews the background and related works for LLM inference. Section III discusses the motivation for the HPIM accelerator. Section IV details the overview of the HPIM. Section V and Section VI illustrate the microarchitecture design of the HPIM and its strategies for workload mapping and execution scheduling, respectively. Section VII presents a comprehensive evaluation of HPIM against state-of-the-art accelerators, and Section VIII concludes the paper.

II. BACKGROUND AND RELATED WORKS

A. LLM Inference

Modern LLMs, predominantly based on the Transformer architecture [26], capture dependencies in text by computing relationships between all pairs of tokens in a sequence. As illustrated in Fig. 2, the architecture consists of an embedding layer for token vectorization, a stack of decoder layers for representation refinement, and a language model (LM) head for probability distribution output. Each decoder layer includes two primary computational blocks: the self-attention mechanism and the feed-forward network (FFN). Specifically, the self-attention mechanism captures contextual relationships
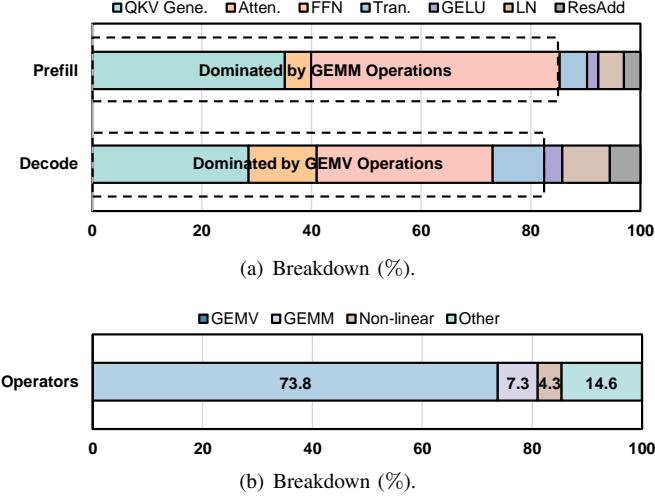
(a) Breakdown (%).

(b) Breakdown (%).

Fig. 3. Execution breakdown of OPT-13B inference on an A100 GPU. (a) Component-level breakdown across the prefill (GEMM-bound) and decode (GEMV-bound) stages. (b) Operator-level breakdown over the entire LLM inference process (e.g., GEMM, GEMV, Softmax). The results are obtained using OPT-13B with an input length of 512 tokens and an output length of 32 tokens. It shows that the overall execution is overwhelmingly dominated by the GEMV-centric decode stage (73.8%), highlighting it as the primary performance bottleneck.

by first projecting token representations into Query (Q), Key (K), and Value (V) matrices via linear transformations. It then computes attention outputs in parallel across multiple attention heads using scaled dot-product attention, defined as:

$$Attention(Q, K, V) = Softmax(\frac{Q \times K^T}{\sqrt{d_k}}) \times V, \quad (1)$$

where d_k is the dimension of the K vectors. The outputs from all attention heads are concatenated and linearly projected to generate the final self-attention result. The FFN typically contains two substantial linear transformations separated by a non-linear activation function (e.g., GELU), accounting for roughly two-thirds of the total model parameters. Each of these blocks is typically preceded by a Layer Normalization (LN) and followed by a residual (ResAdd) connection to ensure numerical stability. The execution of LLM inference on this architecture generally involves two distinct operational phases: prefill and decoding. During the prefill phase, the model processes the entire input sequence in parallel, where the linear transformations within the self-attention mechanism and FFN manifest as computationally intensive General Matrix-Matrix multiplication (GEMM) operations. In the subsequent decoding phase, the model generates tokens sequentially in an autoregressive manner, each prediction dependent on previously generated tokens. To maintain efficiency and avoid redundant computation, the Key and Value vectors for all preceding tokens are stored and reused from a KV cache. This crucial mechanism transforms the repetitive GEMM into a series of memory-bound General Matrix-Vector (GEMV) operations, where the large weight matrices of the self-attention and FFN are multiplied with a single token representation at each step. This fundamental shift from compute-bound GEMM to memory-bound GEMV operations underscores the need for
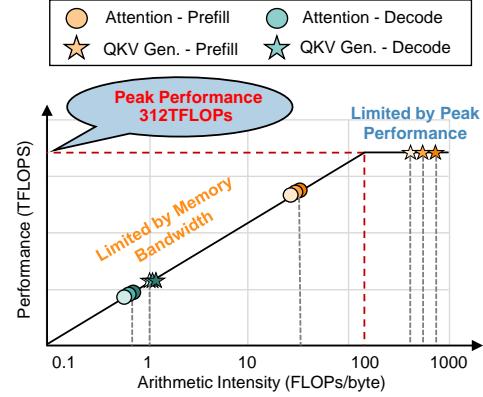

Fig. 4. Roofline model of OPT-6.7B (Bright), OPT-13B (Moderate), and OPT-30B (Dark) operations on an A100 GPU with a sequence length of 2048. The points plot the performance of Attention (circle) and QKV Generation (pentagram) during the prefill (orange) and decode (green) phases.

specialized hardware acceleration tailored specifically to these distinct computational patterns.

B. Traditional Hardware Accelerators for LLM Inference

Traditional hardware accelerators, including GPUs, TPUs, and NPUs, have been extensively adopted for LLM inference due to their mature software ecosystem and optimized computational engines [27, 28]. However, these architectures exhibit inefficiencies when executing single-batch LLM inference, due to the different computational characteristics and bandwidth requirements between the prefill and decoding phases, as shown in Fig. 3(a). While existing platforms are highly optimized for the GEMM-centric prefill phase, their performance is significantly reduced when handling the memory-bound GEMV operations. The analysis in Fig. 3(b) illustrates this imbalance. Even in a workload with a long prompt and short generation (512 input and 32 output tokens), a scenario dominated by the prefill phase, the memory-bound GEMV operations still account for a majority 73.8% of the total execution time. This pronounced inefficiency in handling GEMV consequently positions the decoding phase as the performance bottleneck for LLM inference on conventional accelerators, motivating the urgent need for novel architectural solutions. The roofline model depicted in Fig. 4 further elucidates the intrinsic relationship between arithmetic intensity (FLOPS/byte) and achievable computational performance (TFLOPS) [29]. As shown, the decoding stage in single-batch inference predominantly operates in the memory-bound regime, meaning the performance is predominantly limited by memory bandwidth rather than computing capabilities. With the continued scaling of LLMs, memory bandwidth constraints have emerged as the primary performance bottleneck, reducing throughput and increasing inference latency. Therefore, exploring emerging computing paradigms to accelerate LLM inference has become an important challenge to address.

C. PIM Accelerators for LLM Inference

PIM accelerators have emerged as a promising architectural paradigm for LLM inference, mitigating the memory bandwidth bottleneck for memory-bound GEMV operations in

Arch.	Subsystem for GEMM	Subsystem for GEMV	Inference Focus	Advantages
HPIM (Ours)	SRAM-PIM with PE and Vector Units	Parallel SRAM-PIM and HBM-PIM	Single	Parallel & Hierarchical PIM High Bandwidth Low Latency Computational Diversity
NeuPIMs [30]	NPU	HBM-PIM	Multi	High Throughput
AttAcc [31]	NPU	HBM-PIM	Multi	High Throughput
TransPIM [32]	N/A	HBM-PIM	Single	Monolithic PIM with High Bandwidth
IANUS [33]	NPU	GDDR6	Single	Unified PIM-centric Design
Cambricon-LLM [34]	NPU	Flash	Single	Extreme Capacity, Low Cost
CXL-PNM [22]	PE	LPDDR5X	Single	Scalable & TCO-Efficient

the autoregressive decoding phase. However, different LLM deployment scenarios impose distinct optimization targets, giving rise to a spectrum of PIM architectures. As summarized in Tab. I, these approaches can be broadly categorized into two main classes: high-throughput for multi-batch processing, or low-latency for real-time, single-batch services.

High-Throughput, Multi-Batch Inference Architectures. This class of architectures primarily targets data center scenarios that must handle a large volume of concurrent user requests. The design prioritizes system throughput, aiming to process a maximal volume of requests per unit of time. The prevalent strategy is the adoption of a heterogeneous architecture that assigns GEMM operations to a host processor (e.g., NPU/GPU) while offloading GEMV operations to PIM. For example, NeuPIMs [30] and AttAcc [31] both employ a heterogeneous "NPU+HBM PIM" system, leveraging pipelining and spatial parallelism across multiple requests to maximize hardware utilization and throughput.

Low-Latency, Single-Batch Inference Architectures. In contrast, this class of architectures prioritizes minimizing the end-to-end single-request latency for real-time interactive applications. The predominant strategy remains a heterogeneous architecture, where a host NPU handles GEMM operations in the prefill phase, and a specialized PIM unit accelerates GEMV operations in the decoding phase. Within this paradigm, the primary differentiation among these solutions often lies in the choice of memory technologies and system design. For instance, some leverage LPDDR5X [22] for its unique balance of high capacity and bandwidth, or integrate GDDR6-PIM [33] into a unified memory system with an NPU to reduce data movement. Others utilize Flash-based [34] PIM architecture for its vast capacity and low cost, targeting on-device inference of large LLMs. In contrast to these heterogeneous architectures, TransPIM [32] proposes a monolithic HBM-PIM architecture, employing a software-hardware co-design with novel dataflows to accelerate the entire process.

However, as demonstrated in Sec. II-B, the decoding phase dominates end-to-end latency in single-batch inference. This phenomenon is rooted in the absence of inter-batch parallelism, which forces computation in the decoding phase to be executed serially. Consequently, all the aforementioned architectures that merely accelerate the decoding stage with a single PIM system are insufficient to fully overcome the performance bottleneck in latency-critical scenarios. In response to these identified shortcomings, we propose HPIM, a heterogeneous PIM architecture that integrates tightly-coupled SRAM-PIM and HBM-PIM subsystems. This memory-centric heterogeneous PIM design enables a parallel and pipelined execution fashion for the GEMV operations, directly addressing the serial execution bottleneck.

III. MOTIVATIONS FOR HPIM ACCELERATOR

A. Necessity of Heterogeneous PIM Design

Serial Bottleneck in LLM Inference. As established in Sec. II, single-batch LLM inference is dominated by the autoregressive decoding phase, where each token must be generated strictly after its predecessor. Although prior PIM accelerators alleviate the bandwidth pressure of memory-bound GEMV kernels, they are inherently constrained by this **token-level serial dependency**. To remove this barrier, we must pivot from inter-token to intra-token parallelism, the concurrent execution of sub-tasks within the generation of a single token. This necessitates a composite heterogeneous PIM architecture where different types of PIM subsystems operate in parallel to accelerate these sub-tasks separately. For instance, weight-intensive matrix multiplications can be offloaded to a capacity-oriented unit (e.g., DRAM-PIM) while latency-critical attention computations are assigned to a speed-oriented one (e.g., SRAM-PIM). Such an architectural decomposition enables fine-grained pipelining and parallelism along the critical path of token generation, thereby directly addressing the serial execution bottleneck.

Diversity in Computational Workloads. The inference of LLMs involves diverse computational patterns, including memory-bound GEMV, compute-intensive GEMM operations, and nonlinear operations such as softmax, GELU, and Layer-Norm. These operations exhibit varying demands in terms of memory bandwidth, storage capacity, and compute intensity. Consequently, adopting a homogeneous PIM architecture with a single memory technology proves insufficient. To mitigate this mismatch, a heterogeneous PIM architecture, integrating different memory technologies, is essential to concurrently satisfy the demands for high bandwidth, large storage capacity, and computational diversity, as shown in Fig. 1. For example, SRAM-based PIM offers low-latency computation and high internal bandwidth but suffers from limited capacity; in contrast, DRAM or Flash-based PIM provides large storage but incurs longer access latencies and consumes more energy. Different tasks can be selectively assigned to the most suitable PIM domains based on their computational and memory characteristics. This strategic division improves resource utilization and load balancing.

B. Why HBM and SRAM?

Among various candidate memory technologies, we specifically select HBM and SRAM as the underlying media for our heterogeneous PIM-based accelerator, driven by the following key considerations:

High Bandwidth Requirements. The primary bottleneck of single-batch LLM inference lies in memory bandwidth constraints. Among various memory technologies, HBM and SRAM inherently provide extremely high memory bandwidth.
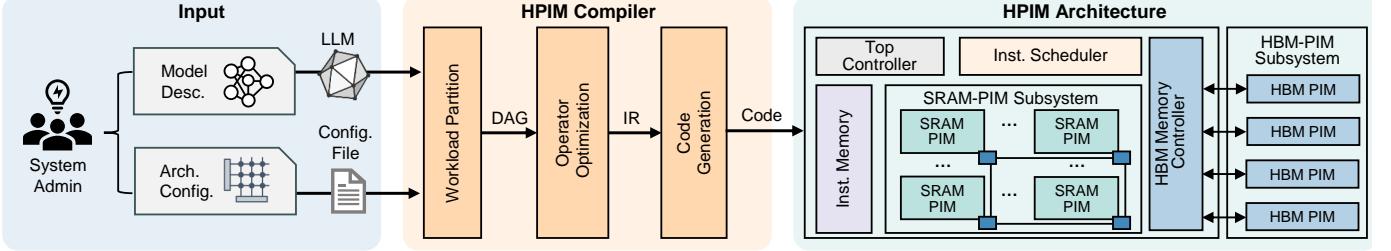
Fig. 5. Overview of the HPIM accelerator, including its two main components: HPIM compiler and HPIM architecture. The workflow starts from user-defined model descriptions and architectural configurations, which are processed by the HPIM compiler to generate executable instructions. These instructions are then run on the HPIM architecture, which features a tightly-coupled design of an SRAM-PIM subsystem and an HBM-PIM subsystem.

Moreover, integrating compute logic in memory banks or arrays enables even higher internal bandwidth utilization, substantially alleviating bandwidth-related performance bottlenecks.

Cost Efficiency and Integration Feasibility. Despite its relatively recent emergence, SRAM-based PIM architectures have witnessed rapid development due to their excellent compatibility with standard digital processes. This compatibility facilitates the seamless integration of full-core digital logic with SRAM-based PIM, enabling efficient implementation of complex operations, such as nonlinear activation functions. Conversely, embedding such intricate logic directly into DRAM or emerging non-volatile memory (NVM) technologies remains prohibitively challenging and costly. In such media, only simpler operations (e.g., multiply-accumulate, MAC) are realistically feasible. This delineation naturally positions SRAM as an optimal candidate for computationally intensive tasks requiring intricate logic integration.

Reduction of Data Movement Overhead. The weight matrices constitute a dominant fraction compared to the KV cache in single-batch inference scenarios. For instance, in an OPT-30B model with a $1k$ sequence length, the weights amount to approximately 60 GB, while the KV cache is only about 1.31 GB. Extensive prior work demonstrates that frequent data movement is one of the primary sources of energy dissipation and performance loss. Consequently, computations involving substantial weight matrices in QKV generation and FFN are strategically offloaded to HBM-based near-bank PIM to capitalize on the larger storage capacity and bandwidth advantages of HBM. In contrast, smaller-sized but computationally intensive operations, particularly attention computations, and nonlinear operations, are effectively mapped onto SRAM-based PIM units, leveraging their unique characteristics of ultra-high speed, low latency, and high throughput computation.

In summary, combining HBM-PIM with SRAM-PIM provides a highly practical, performance-advantaged architecture for single-batch LLM inference. HBM offers the bandwidth and capacity required for weight-intensive GEMV computations, while SRAM delivers ultra-low latency for attention and nonlinear operators. Our heterogeneous PIM design minimizes data movement, enables fine-grained pipelining and parallel execution, simultaneously reduces end-to-end latency, and provides implementation feasibility.

IV. OVERVIEW OF THE HPIM ACCELERATOR

To address the critical bottlenecks in single-batch LLM inference, particularly the memory bandwidth limitations and serial dependencies in autoregressive decoding, we propose HPIM, a memory-centric heterogeneous PIM accelerator. As depicted in Fig. 5, HPIM comprises two tightly integrated components: a hardware-aware compiler framework and a heterogeneous hardware architecture.

A. HPIM Compiler Framework

The compiler framework bridges high-level LLM models with the underlying heterogeneous PIM architecture. It takes as input the model graph and the system configuration, and performs a series of hardware-aware transformations to generate optimized PIM instructions. First, the compiler conducts operator analysis and annotation, tagging each node in the LLM graph based on its computational and memory characteristics (GEMV, GEMM, or nonlinear, etc.). Recognizing distinct computational patterns, the compiler adopts a stage-specific operator mapping policy:

1) *Prefill Stage.* All computations, primarily large-scale GEMM kernels, are entirely dispatched to the SRAM-PIM subsystem. This leverages the powerful processing capabilities of the SRAM-PIM subsystem, particularly its specialized compute units optimized for GEMM-intensive workloads.

2) *Decoding Stage.* The compiler strategically partitions the workload between the SRAM-PIM and HBM-PIM subsystems, enabling concurrent execution across these heterogeneous modules. This balanced distribution effectively reduces pipeline stalls, minimizing serial dependencies.

Then, the compiler applies a hybrid tiling strategy, combining head-wise parallelism (HP) for multi-head attention and tensor-wise parallelism (TP) for spatially partitioning large matrices. This strategy effectively balances computational workloads and minimizes inter-subsystem data movement. Subsequently, the optimized computation graph is translated into an intermediate representation (IR), capturing optimized operator mapping and execution schedules. This IR is finally lowered into separate PIM-specific instruction streams for SRAM-PIM and HBM-PIM subsystems, including synchronization, data prefetching, and pipeline control instructions.
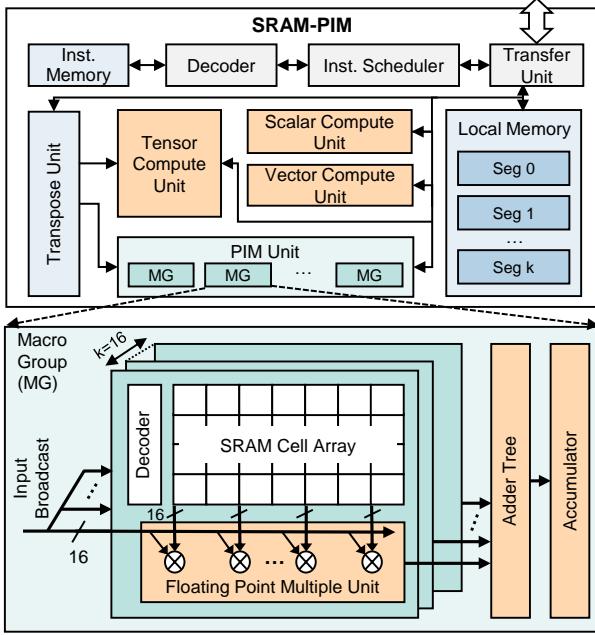
Fig. 6. The microarchitecture of the SRAM-PIM subsystem, which is designed as a computing engine (Tensor, Vector, Scalar, and PIM Units) within the HPIM architecture.

B. HPIM Architecture

The HPIM architecture comprises an SRAM-PIM subsystem, an HBM-PIM subsystem, an HBM memory controller, a top controller, a hardware instruction (Inst.) scheduler, and on-chip Inst. memory. The SRAM-PIM Subsystem is built upon a multi-core architecture. It comprises 32 individual SRAM-PIM cores, facilitating high computational density and tightly-coupled high-bandwidth memory. These cores, each with an independent instruction control flow, are interconnected by a high-throughput Network-on-Chip (NoC). This organization facilitates scalable parallel execution across multiple attention heads or tensor matrices and enables flexible inter-core pipelining. HBM-PIM Subsystem consists of four HBM-PIMs, which function as high-capacity data storage and high-bandwidth parallel processors. Each HBM-PIM stores large model weights and the KV cache required for generative inference. Lightweight arithmetic circuits are embedded at the DRAM-bank level so that multiplications and accumulations are performed close to the data while reducing area and power overhead. Meanwhile, to facilitate seamless data flow between the two subsystems, each SRAM-PIM core is equipped with a dedicated interface that provides direct access to specific HBM channels, enabling efficient, direct data transfers. Furthermore, a centralized top-level controller combined with a hardware-based instruction scheduler dynamically orchestrates task execution, optimally pipelines computations across SRAM and HBM units, and efficiently synchronizes data movement.

In summary, by intelligently partitioning tasks based on hardware advantages, employing a sophisticated compiler strategy, and leveraging deep pipelining between heterogeneous subsystems, HPIM significantly reduces inference latency and maximizes throughput in single-batch LLM inference workloads.

V. MICROARCHITECTURE DESIGN OF HPIM

This section elaborates on the microarchitectural details of the two subsystems outlined in §VI-B.

A. SRAM-Based PIM Design

As shown in Fig. 6, the SRAM-PIM subsystem functions as a high-performance computing engine composed of computational units, storage modules, and a sophisticated instruction scheduling module.

Computational Module. Each SRAM-PIM core incorporates multiple specialized computational units to meet the different computation requirements in LLM inference. To achieve high internal bandwidth, this unit performs computation directly within SRAM arrays. Each unit consists of 16 macro groups (MGs) that share the same inputs and are equipped with integrated floating-point multipliers, adder trees, and accumulators. However, the PIM unit solely performing GEMV operations is insufficient for complete LLM inference. To handle the compute-intensive GEMM operations prevalent in the prefill stage, each core includes a powerful tensor Compute Unit (TCU), which is realized by a 64×64 systolic array for high-throughput matrix multiplication. Concurrently, a programmable vector compute Unit (VCU) is responsible for essential element-wise operations. It supports a range of scalar and vector-level operations, including addition, subtraction, multiplication, division, exponential, and square root. These capabilities are crucial for implementing complex non-linear functions like Softmax and GELU, normalization layers like LayerNorm, and residual additions. The scalar compute unit (SCU) focuses on executing scalar arithmetic and managing control flow operations.

Instruction Scheduling Module. To support diverse workloads and simplify compiler design, the SRAM-PIM core implements a unified 32-bit instruction set architecture (ISA) with specialized variants for different operation types. Instructions are categorized into three main classes: compute, communication, and control flow instructions. The compute instructions are further specialized for the PIM unit, TCU, VCU, and SCU. The execution of this instruction set is managed by a three-stage pipeline (Fetch-Decode-Execute) designed to maximize throughput. The process begins at the Fetch stage, where compiler-generated instructions are retrieved from instruction memory. During the subsequent Decode stage, each instruction is parsed and dispatched to a dedicated instruction scheduler. This scheduler is central to the Execute stage, where it dynamically analyzes data dependencies, monitors the status of the various compute units, and orchestrates task execution to maintain high pipeline utilization.

Storage Module. The storage module encompasses local memory for input/output data caching, a dedicated transpose unit, and a transfer unit. The transpose unit efficiently transforms the K matrix retrieved from DRAM into an appropriate format for subsequent computation within the PIM unit. The transfer unit facilitates efficient data movement between various computational and storage modules within the SRAM-PIM subsystem, ensuring streamlined communication and data accessibility.
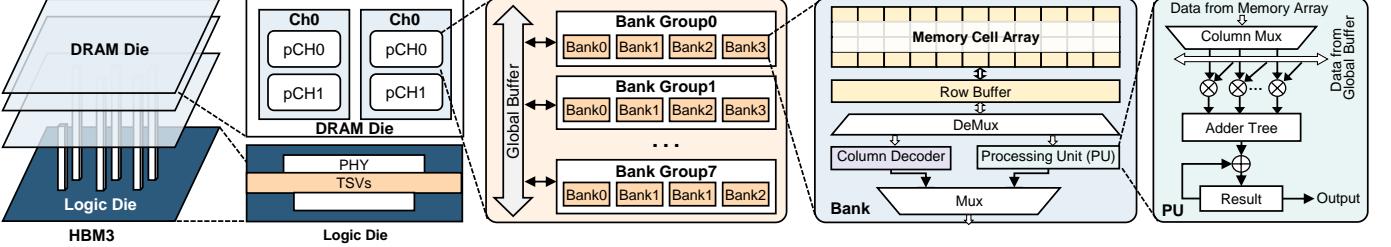
Fig. 7. The microarchitecture of the HBM-PIM subsystem. It illustrates the hierarchical structure based on HBM3, from the 3D-stacked package down to the the in-bank Processing Unit (PU).

Overall, these tightly integrated components substantially improve the efficiency, scalability, and overall performance of the SRAM-PIM subsystem within the HPIM architecture.

B. HBM-Based PIM Design

The HBM-PIM subsystem, illustrated in Fig. 7, leverages an advanced HBM3 stack architecture comprising 8 vertically stacked 8-Hi DRAM dies interconnected by TSVs. Each DRAM die contains 2 independent channels, each with 2 pseudo-channels (pCH). These pCHs are further subdivided into 8 bank groups (BG) and 4 banks, facilitating massive internal bandwidth and highly parallel data access. To avoid the prohibitive cost of integrating complex, full-core logic into HBM, our HBM-PIM subsystem adopts a simplified compute logic integrated at the bank level. Building upon previous PIM paradigms (e.g., Newton architecture [35]), each bank integrates dedicated multiply-accumulate (MAC) units to process large-scale, weight-intensive GEMV computations directly near the memory cell arrays, significantly reducing data movement overhead. In contrast to the SRAM-PIM subsystem, the HBM-PIM units are managed by a set of simple commands rather than a full-fledged ISA. This design choice minimizes control overhead and hardware complexity within the bank.

The DRAM die supports two operational modes: a standard read/write mode and a dedicated compute mode. In standard read/write mode, data is transferred between the memory array and external interfaces. In compute mode, activation is initially loaded into the global buffer shared among all banks within a pCH, and then simultaneously broadcast to each bank. Concurrently, weight and activation streams are processed in parallel by GEMV units, efficiently performing multiplication and accumulation operations. To further minimize unnecessary data movement, our design fully exploits input reuse by ensuring input elements across matrix rows are loaded only once. The partial sums resulting from computations are temporarily stored within dedicated result units, which subsequently transfer final computed results either externally or write them back into the HBM storage. This design strategy significantly enhances computational efficiency by optimizing data locality and maximizing parallelism within the HBM-PIM subsystem.

In summary, the proposed heterogeneous architecture of HPIM efficiently partitions computation across SRAM-PIM and HBM-PIM subsystems based on latency sensitivity and bandwidth demands. SRAM-PIM emphasizes low-latency, versatile computation for GEMM, GEMV, and nonlinear operations, while HBM-PIM focuses on maximizing internal memory bandwidth for weight-intensive GEMV operations. Coordinated by a centralized instruction scheduler and optimized by the compiler, this synergistic design significantly enhances performance, efficiency, and scalability for single-batch LLM inference.

VI. HARDWARE-AWARE SCHEDULING OF HPIM

This section focuses on the hardware-aware scheduling of HPIM systems. We first elaborate on the workload mapping techniques of the weight matrices and attention layers between the heterogeneous HBM-PIM and SRAM-PIM subsystems. Then, we detail the scheduling strategy derived from these mappings, focusing on pipeline design tailored for multi-head self-attention, load balancing considerations, and coordinated optimization across prefetching, computation, and data transmission tasks.

A. Workload Mapping

Modern LLMs exhibit significant variability in the number of attention heads, typically ranging from 16 to 128. To maximize resource utilization and improve the scalability of the various LLM models, we adopt a hybrid parallelism strategy that combines HP and TP across both the HBM-PIM and SRAM-PIM subsystems.

On the HBM-PIM side, the weight matrices of Q, K, and V employ this hybrid parallelism strategy, as depicted in Alg. 1. Specifically, the weight matrices are first partitioned along the head dimension, assigning each head compute on one or more dedicated HBM channels. Then, we utilize column-wise tiling coupled with channel-wise interleaving across multiple HBM channels to maximize memory throughput. This strategy ensures that each DRAM-PIM channel independently manages distinct, fine-grained partitioned weight matrices to fully exploit available internal memory bandwidth. For weights associated with other FC layers (e.g., weight1, weight2, and weight3 in Fig. 5), we leverage TP by slicing large weight matrices along spatial or row dimensions. These matrices are then mapped across multiple HBM channels, significantly reducing data movement, balancing computational loads, and matching the access granularity of DRAM banks.

Fig. 8 demonstrates the detailed mapping procedure using the K weight matrix with 16 heads as an example. This
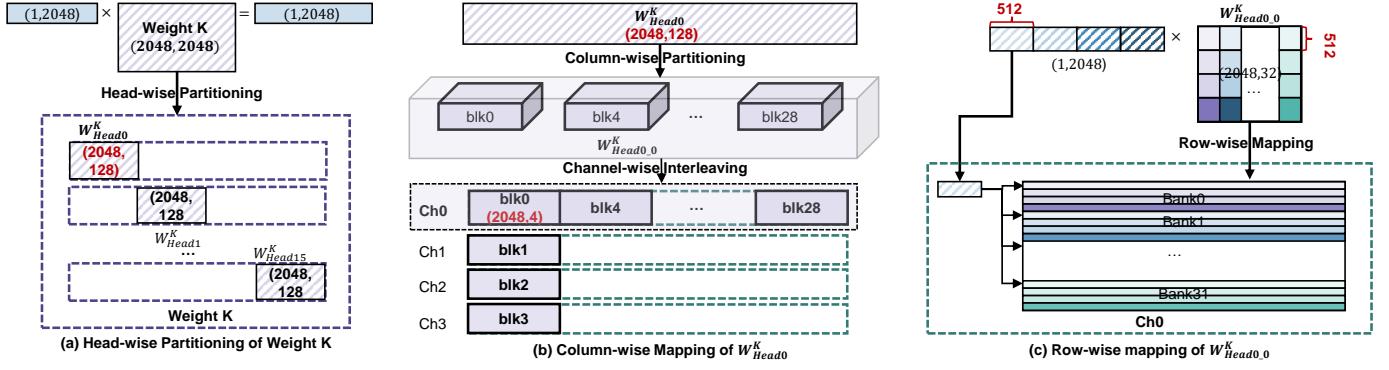
Fig. 8. Illustration of weight K of 16 heads allocation and tiling scheme for HBM-PIM subsystems.

Algorithm 1 Hybrid Parallelism of Q/K/V Weight Allocation

Require: Number of heads N_h, DRAM channels N_D, SRAM cores N_S, weight dimensions d_{emb}, tensors $\{T_h\}$ for each head h.

Ensure: Partitioned tensor slices $\{S_{c,r}\}$ to each channel c and round r.

1: // Calculate dimensions per head
2: $d_k \leftarrow d_{emb}/N_h$
3: // Initialize counters for allocated heads and rounds
4: $h_{idx} \leftarrow 0$
5: $r \leftarrow 0$
6: **while** $h_{idx} < N_h$ **do**
7:      // Calculate the number of remaining heads
8:      $h_{rem} \leftarrow N_h - h_{idx}$
9:      **if** $h_{rem} <= 0$ **then break**
10:     **end if**
11:     // Calculate the number of heads for HP in this round
12:     $h_r \leftarrow \min(h_{rem}, N_D, N_S)$
13:     $h_p \leftarrow 2^{\lfloor \log_2 h_r \rfloor}$
14:     // Calculate allocated channels per head for TP
15:     $N_{ch} \leftarrow N_D/h_p$
16:     **for** $h = h_{idx}$ to $h_{idx} + h_p - 1$ **do**
17:         // Apply channel-wise interleaving
18:         **for** $i = 0$ to d_k **do**
19:             $c \leftarrow (h - h_{idx}) \cdot N_{ch} + (i\%N_{ch})$
20:             $S_{c,r} \leftarrow \text{Append}(T_h[i])$
21:         **end for**
22:     **end for**
23:     $h_{idx} \leftarrow h_{idx} + h_p$
24:     $r \leftarrow r + 1$
25: **end while**

---

matrix is first partitioned head-wise into smaller sub-matrices, enabling head-wise parallelism. These sub-matrices are then tiled along the column dimension and allocated across multiple HBM channels through an interleaving distribution scheme to exploit tensor-wise parallelism. Finally, individual blocks are row-wise mapped onto DRAM banks within each channel to maximize bank-level throughput and align with the native access granularity of HBM. This multi-level mapping scheme ensures scalable and internally bandwidth-efficient weight loading for attention computations.

On the SRAM-PIM side, we adopt a combination of HP and intra-head TP to accommodate varying model configurations efficiently. Each head can be executed either independently on a single core or collaboratively across multiple cores to maximize parallelism and resource utilization. For example, when the number of heads matches the number of available compute cores (e.g., 32 heads on 32 cores), each head is exclusively mapped to a single core. This arrangement allows entirely localized attention computation, including softmax normalization, eliminating inter-core communication and minimizing latency. For larger models with more heads than available cores, computations are partitioned and scheduled across multiple execution phases.

Conversely, when the number of heads is fewer than the available compute cores, we apply intra-head TP by evenly distributing the computation of a single head across multiple cores. As illustrated in Fig. 9, each core handles a distinct tensor partition and computes partial results independently. Then, we apply an efficient All-Gather to recover the global normalization factors for softmax. This strategy preserves computation locality while maintaining correctness with minimal NoC overhead.

B. Pipelined Parallelism for LLM Inference

During the prefill phase, the workload is dominated by large-scale GEMM kernels, as shown in Fig. 3(a). Due to its superior arithmetic throughput for GEMM kernels, the TCU in the SRAM-PIM core is designated to execute all such operations, including Q, K, and V generation, the attention projection, and the subsequent feed-forward network (FFN) layers. This allocation ensures that the most compute-intensive components of the model are executed on the most capable compute unit, thereby maximizing performance efficiency. Element-wise non-linear kernels such as layer normalization, residual addition, GELU, and the softmax are executed in the VCU. As illustrated in Fig. 10(a), the detailed execution can be described as follows:

- **Weight Prefetching and Loading.** Initially, weight matrices such as W_i^K are streamed from HBM-PIM into the SRAM-PIM subsystem, and subsequently, prefetching of subsequent weights (W_i^Q, W_i^V, etc.) is conducted.
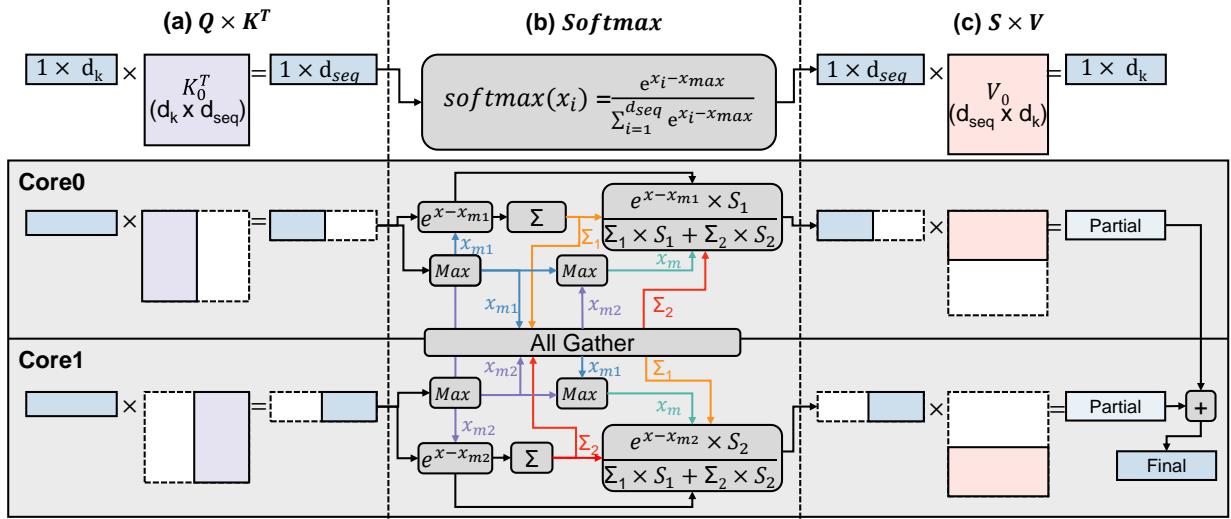
Fig. 9. Illustration of parallel processing of a single head on multiple SRAM-PIM cores. The figure illustrates the three main stages of attention: (a) GEMV operation between the query vector and the transposed key matrix ($Q \times K^T$), (b) softmax computation with distributed max and sum operations across cores and subsequent All-Gather synchronization, and (c) GEMV operation of the attention scores with the value matrix ($S \times V$).

- **Computation-Intensive GEMM in TCU.** Upon receiving weight matrices from the HBM-PIM subsystem, the TCU initiates computation-intensive GEMM operations required for QKV generation and attention computations. Initially, the K_i matrix is computed, followed promptly by the generation of matrix Q_i. During the computation of Q_i, the transpose operation concurrently converts K_i into K_i^T, enabling immediate initiation of the $Q_i \times K_i^T$ computation. The output of this GEMM is pipelined directly into VCU for the softmax operation. Simultaneously, weights for the value matrix (W_i^V) are prefetched, facilitating the immediate computation of the subsequent $S \times V$ operation. This pipeline strategy minimizes computational stalls and maximizes throughput.

- **Tail Processing and FC Layers Execution.** Upon completing all attention heads, tail computations, including projection, residual addition, LayerNorm, and GELU activation, are sequentially executed. Weight matrices associated with FC layers are uniformly distributed across TCU, while other computations are allocated within the VCU of the SRAM-PIM subsystems. This distribution strategy ensures balanced workloads and optimal resource utilization, efficiently concluding the prefill stage without pipeline stalls.

During the decoding phase, the main workload shifts toward memory-bound GEMV operations, and memory bandwidth becomes the primary bottleneck. To alleviate the bandwidth limitation, weight-intensive FC layers are migrated to the HBM near-bank PIM subsystem, while latency-critical attention kernels and non-linear kernels remain in the SRAM-PIM subsystem. Since $Q \times K^T$ and $S \times V$ are also GEMV operations, we deliberately execute them in the PIM Unit of the SRAM-PIM subsystem to enable the computation pipeline. As shown in Fig. 10(b), the decoding execution can be broken down as follows:

- **Bandwidth-Critical GEMVs in HBM-PIM.** All weight-intensive GEMV computations, including $Q/K/V$ generation and FFN layers, are executed within near-bank HBM-PIM units. Each channel computes the K, Q, and V vectors for its assigned heads sequentially and streams them to the corresponding SRAM-PIM cores.

- **Pipelined Attention.** To minimize idle time of execution, SRAM-PIM subsystems are tightly pipelined with HBM-PIM execution. For example, once the K vector is generated and streamed into the SRAM-PIM core, the transpose unit immediately begins converting K into K^T in parallel with the ongoing generation of Q in HBM-PIM. Likewise, the computation of $Q \times K^T$ in PIM Unit is overlapped with the generation of V. This tightly coupled pipelining between SRAM-PIM and HBM-PIM execution shortens the critical path and improves throughput. If multiple cores are assigned to a single head, we perform an all-gather softmax reduction, where only local maxima and exponent sums are exchanged to minimize NoC traffic. The resulting attention scores are forwarded back to the PIM units for the $S \times V$ GEMV, while the HBM-PIM fabric concurrently computation and transforms the next head group.

- **Projection and FFN Distribution.** After attention, the subsequent projection and FFN layers, which involve weight-heavy GEMV operations, are still offloaded to the near-bank HBM-PIM subsystem. The corresponding weight matrices are pre-partitioned and striped across HBM channels to enable parallel execution. This channel-aligned mapping ensures load balancing across memory stacks.

VII. Evaluation Results

A. Experimental Setup

LLM Models. We evaluate a series of transformer-based LLMs from the OPT family [36], with configurations ranging from 350M to 30B parameters, as detailed in Tab. II. This
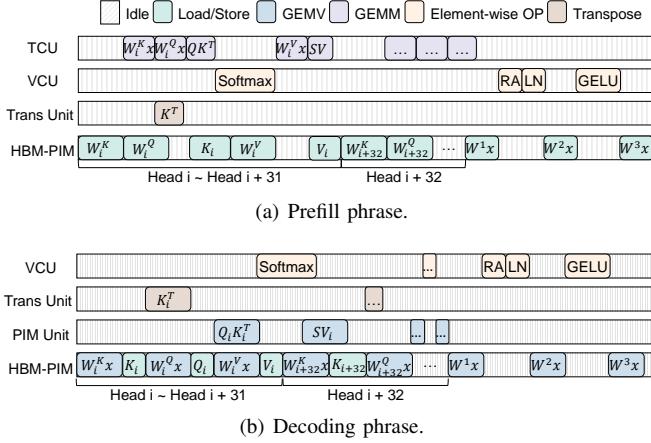
(a) Prefill phrase.

(b) Decoding phrase.

Fig. 10. Example execution timelines of 32-head for (a) prefill phrase, where FCs are mapped to the TCU, and for decoding phrase, where FCs are mapped to the PIM. TCU: tensor compute unit, VCU: vector compute unit, Trans Unit: transpose unit, PIM Unit: PIM unit in SRAM-PIM core.

allows us to explore the scalability and efficiency of HPIM across diverse LLM configurations. All baseline models are executed using FP16 precision under a single-batch (batch size = 1) inference setting. While our evaluation focuses on OPT as representative benchmarks, our proposed HPIM architecture is model-agnostic and can be readily extended to other transformer-based LLMs.

Baseline. We evaluate HPIM against a set of state-of-the-art baselines, including the NVIDIA A100 GPU, and two representing PIM-related accelerators: IANUS hybrid NPU-PIM system [33] and the CXL-PNM near-memory [22] platform. All baseline workloads are executed using the Huggingface Transformers framework [37].

Hardware Specification of HPIM Architecture. The hardware configuration of our proposed HPIM architecture is detailed in Tab. IV. The system integrates 32 SRAM-PIM cores with 4 HBM-PIM modules. Each SRAM-PIM core is a powerful processing engine containing several specialized compute units (TCU, VCU, SCU, PIM Unit) and local memory. The HBM-PIM subsystem is built upon an HBM3-based PIM design.

HPIM Simulator. We evaluate the proposed HPIM architecture using a cycle-accurate simulator capable of capturing detailed compute and memory behaviors across both the SRAM-PIM and HBM-PIM subsystems. The HBM-PIM subsystem is modeled using an extended version of the open-source DRAMsim3 simulator [38]. This extensive simulator supports bank-level compute execution and PIM-aware scheduling, with timing and power parameters configured for HBM3 memory [39]. Meanwhile, we design a customized simulator for the SRAM-PIM subsystem. The compute logic for each functional unit within this subsystem is described in Verilog HDL. Memory-related characteristics are extracted from the memory compiler.

B. Performance Evaluation

End-to-End Inference Latency. We first evaluate the end-to-end inference latency and speedup of our proposed HPIM

architecture compared to an NVIDIA A100 GPU. As illustrated in Fig. 11, HPIM consistently achieves significantly lower inference latency across all tested model scales and sequence lengths, delivering a peak speedup of up to $34.3\times$. For the LLM inference workload with significantly more output tokens than input tokens, i.e., (256,768), HPIM demonstrates $4.6\times$, $3.7\times$, and $3.9\times$ lower latency than the A100 GPU for the OPT-6.7B, OPT-13B, and OPT-30B models, respectively. This performance gain is particularly significant considering that the peak compute throughput of HPIM (65 TFLOPS for HBM-PIM and 131 TFLOPS for PIM unit in SRAM-PIM) is substantially lower than that of the A100 GPU (312 TFLOPS). This result underscores a fundamental insight: single-batch inference performance is dictated by architectural efficiency in the memory-bound decoding stage, not just peak compute throughput. By unlocking intra-token parallelism, HPIM concurrently executes different GEMV tasks across its SRAM-PIM and HBM-PIM subsystems. This memory-centric, parallel approach directly mitigates the data movement and serial dependency bottlenecks that constrain conventional accelerators, demonstrating the superiority of its design for

TABLE II
LLM CONFIGURABLE PARAMETERS.

Models		Embedding dimension	Layers	Heads	Params	d_k
OPT	350M	1024	24	16	350M	64
	1.3B	2048	24	32	1.3B	64
	6.7B	4096	32	32	6.7B	128
	13B	5120	40	40	13B	128
	30B	7168	48	56	30B	128

TABLE III
SPECIFICATIONS OF A100, LANUS, CXL-PNM AND HPIM.

	A100	LANUS [33]	CXL-PNM [22]	HPIM
Frequency (MHz)	1155	700	1024	1024
Throughput (TFLOPS)	312	184	4.09	262
SRAM Capacity (MB)	84	64	64	45
DRAM Type	HBM2e	GDDR6	LPDDR	HBM3
DRAM Capacity (GB)	80	8	512	96
DRAM Bandwidth (GB/s)	1935	256	1100	3276
DRAM Internal BW (TB/s)	N/A	4	1.1	102.4

TABLE IV
HARDWARE CONFIGURATION OF HPIM.

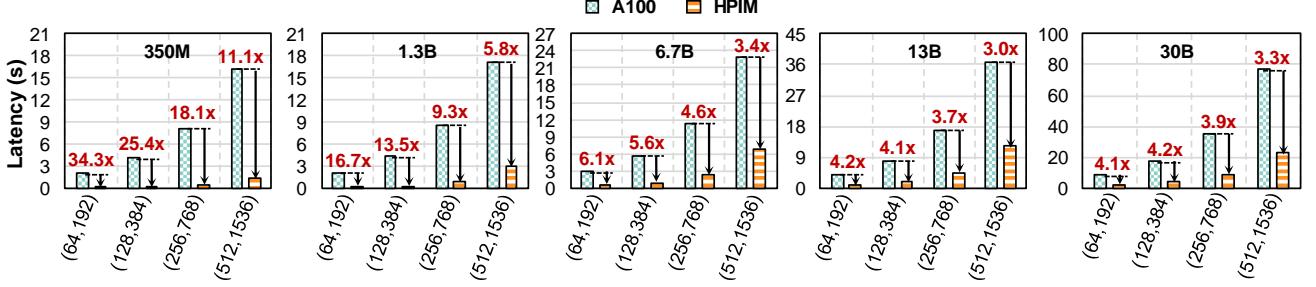|	Composition	4 HBM-PIM, 32 SRAM-PIM, 1GHz Frequency
SRAM-PIM	TCU	64*64 PEs, 1 MAC per PE
	VCU	Add, Sub, Mult, Div, Exp, Square Root, etc.
	PIM Unit	16 MG, 16 SRAM-PIM Macro per MG, 4KB per SRAM-PIM Macro, 8 FP16 Multipliers per Macro, 4.09 TFLOPS
	Local Memory	384KB Activation Memory 32KB Temp Memory
HBM-PIM	HBM3 Configuration	8 Die/HBM, 24Gb/die, 8 Hi-DRAM, 2 Channels per DRAM, 8 Bank Group (BG) per Channel, 4 Bank per BG, 1KB Page Size per pCH
	Processing Unit (PU)	1GHz, 1 PU per Bank, 16 FP16 Multipliers per PU
	Global Buffer	One 1 KB Global Buffer per pCH

Fig. 11. End-to-end inference latency comparison between HPIM and an NVIDIA A100 GPU across a range of OPT models (350M to 30B) and sequence lengths.
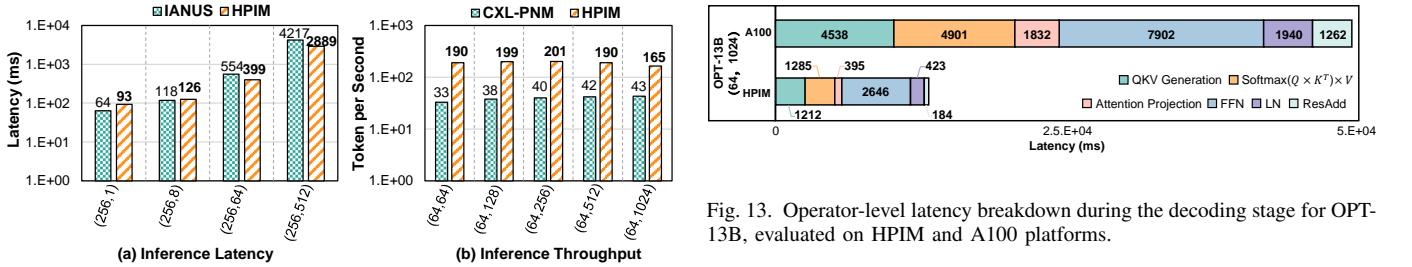

Fig. 12. Performance comparison of HPIM against SOTA accelerators IANUS and CXL-PNM on the OPT-13B model. (a) End-to-end inference latency comparison with IANUS. (b) Throughput comparison with CXL-PNM.

latency-critical applications.

Comparison with Other SOTA Works. To demonstrate the superiority of our architecture, we benchmark HPIM on the OPT-13B model against both the IANUS and the CXL-PNM platforms. For a fair comparison, we maintain identical input/output token configurations as reported in prior work [22, 33]. To accommodate the memory requirements of the 13B model, the IANUS system utilizes four devices interconnected via a PCIe 5.0×16 host interface.
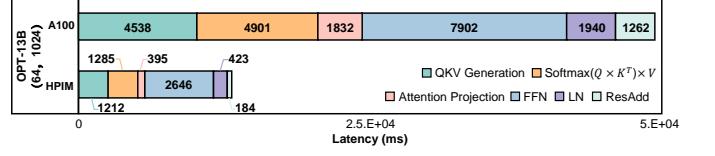
As depicted in Fig. 12(a), the end-to-end performance dynamics change with the sequence length. For short-sequence workloads, such as (256, 1) and (256, 8), HPIM exhibits a slightly higher latency than the IANUS system. This is because for short-sequence workloads, the overall latency is dominated by the prefill stage, which consists primarily of compute-heavy GEMM operations. In this phase, the 4-device IANUS system provides superior compute throughput, giving it a slight performance advantage. Conversely, for longer sequences where the memory-bound decoding phase dominates the workload, HPIM's architectural advantages become pronounced, resulting in significantly lower end-to-end latency. This trend culminates in the (256, 512) configuration, where HPIM completes the task in only 2.89s compared to IANUS's 4.22s, achieving a $1.50\times$ speedup. This significant performance improvement is attributed to three core architectural innovations. First, the HBM3-PIM subsystem of HPIM offers substantially higher memory bandwidth than the GDDR6-PIM of IANUS. Second, HPIM offloads latency-critical attention GEMVs to its specialized PIM unit in SRAM-PIM core. This is architecturally superior to IANUS's approach of using an NPU, which is often underutilized and inefficient for such inherently memory-bound operations. This advantage

Fig. 13. Operator-level latency breakdown during the decoding stage for OPT-13B, evaluated on HPIM and A100 platforms.

becomes more pronounced when processing long sequences, where the execution time of the attention module grows significantly. In such scenarios, replacing the conventional NPU with SRAM-PIM for attention computation offers clear benefits. Third, HPIM's monolithic scale-up design entirely avoids the significant communication overhead inherent in a multi-node scale-out system like IANUS, where inter-device data synchronization over PCIe becomes a critical bottleneck.

This trend of architectural superiority continues when comparing throughput against CXL-PNM. Fig. 12(b) illustrates that HPIM delivers substantially higher tokens per second (TPS), achieving a peak of up to $5.76\times$ over CXL-PNM. This advantage is rooted in HPIM's true in-memory computing paradigm, which harnesses the massive internal bandwidth of HBM3. In contrast, CXL-PNM, a near-memory solution, remains constrained by the limited bandwidth of its external CXL interconnect. Overall, these results demonstrate the effectiveness of HPIM's heterogeneous, fully PIM-centric architecture. By mitigating the memory bottleneck directly at its source, HPIM outperforms both hybrid NPU-PIM and near-memory accelerators, highlighting the significant benefits of a dedicated, memory-centric design for large language model inference.

Layer-Wise Breakdown Analysis. To quantify the performance gains of HPIM over the baseline A100, we analyze the operator-level latency breakdown of OPT-13B in the decoding stages with $1K$ output lengths, as illustrated in Fig. 13. Our analysis shows that HPIM's heterogeneous subsystems effectively accelerate different components of the workload. First, weight-intensive GEMV operations such as QKV generation, attention projection, and FFN layers are offloaded to the HBM-PIM subsystem. By leveraging HBM's high internal bandwidth and near-bank computation to eliminate off-chip data movement, HPIM reduces the execution times for these layers from 4538ms, 1832ms, and 7902ms down to 1212ms, 395ms, and 2646ms, respectively. This corresponds

to significant speedups of $3.74\times$, $4.64\times$, and $2.99\times$ over the A100 baseline. Concurrently, the latency-critical attention computations ($softmax(Q \times K^T) \times V$) are mapped to the SRAM-PIM subsystem. The PIM units in SRAM-PIM offer high internal bandwidth and near-array computing capabilities, effectively accelerating the core attention GEMV operations ($Q \times K^T$, $S \times V$). Crucially, due to the deep pipelining between HBM-PIM and SRAM-PIM subsystems, the execution of the attention computation on SRAM-PIM is effectively masked by the concurrent operations on HBM-PIM. Through this approach, the effective latency of the attention task on the SRAM-PIM side is reduced to $1285ms$, achieving a $3.81\times$ speedup for this portion of the workload. In summary, this combination of targeted partition strategy and system-level pipelined parallelism results in a comprehensive $3.64\times$ end-to-end speedup over the baseline A100.

VIII. CONCLUSION

This paper introduces HPIM, a novel memory-centric heterogeneous Processing-In-Memory accelerator designed to address the critical performance bottlenecks in single-batch LLM inference. HPIM integrates an HBM-PIM subsystem for weight-intensive GEMV operations and an SRAM-PIM subsystem for latency-critical attention computations. This hardware-aware partitioning, combined with a deep pipeline strategy, unlocks intra-token parallelism to directly mitigate the serial dependency inherent in autoregressive decoding. Compared to a state-of-the-art NVIDIA A100 GPU, HPIM achieves a peak speedup of up to $34.3\times$. Furthermore, HPIM outperforms other state-of-the-art PIM-based accelerators, showing a $1.50\times$ speedup over IANUS and up to $5.76\times$ higher throughput than CXL-PNM. These results validate that by embracing a heterogeneous, memory-centric design, HPIM provides a practical and high-performance path forward for accelerating LLM model inference.

REFERENCES

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language Models are Few-shot Learners," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[2] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language Models are Unsupervised Multitask Learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[3] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The Llama 3 Herd of Models," *arXiv preprint arXiv:2407.21783*, 2024.

[4] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "LLaMA: Open and Efficient Foundation Language Models," *arXiv preprint arXiv:2302.13971*, 2023.

[5] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "LLAMA 2: Open Foundation and Fine-Tuned Chat Models," *arXiv preprint arXiv:2307.09288*, 2023.

[6] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez, "Gorilla: Large Language Model Connected with Massive APIs," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[7] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[8] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[9] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[10] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering," in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[11] Y. Zhang, H. Ruan, Z. Fan, and A. Roychoudhury, "AutoCodeRover: Autonomous Program Improvement," in *Proceedings of International Symposium on Software Testing and Analysis (ISSTA)*, 2024.

[12] R. Guo, L. Wang, X. Chen, H. Sun, Z. Yue, Y. Qin, H. Han, Y. Wang, F. Tu, S. Wei *et al.*, "A 28nm 74.34 TFLOPS/W BF16 Heterogenous CIM-Based Accelerator Exploiting Denoising-Similarity for Diffusion Models," in *Proceedings of International Solid-State Circuits Conference (ISSCC)*, 2024.

[13] X. Fu, J. Yue, M. Faizan, Z. Li, Q. Huo, and F. Zhang, "SHMT: An SRAM and HBM Hybrid Computing-in-Memory Architecture With Optimized KV Cache for Multimodal Transformer," *Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, 2025.

[14] C. Duan, J. Yang, X. He, Y. Qi, Y. Wang, Y. Wang, Z. He, B. Yan, X. Wang, X. Jia, and W. Zhao, "DDC-PIM: Efficient Algorithm/Architecture Co-Design for Doubling Data Capacity of SRAM-based Processing-in-Memory," *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 43, no. 3, pp. 906–918, 2024.

[15] C. Duan, J. Yang, Y. Wang, Y. Wang, Y. Qi, X. He, B. Yan, X. Wang, X. Jia, and W. Zhao, "Towards Efficient SRAM-PIM Architecture Design by Exploiting Unstructured Bit-Level Sparsity," in *Proceedings of Design Automation Conference (DAC)*, 2024.

[16] Y. Qi, J. Yang, Y. Wang, Y. Wang, D. Wang, L. Tang, C. Duan, X. He, and W. Zhao, "CIMFlow: An Integrated Framework for Systematic Design and Evaluation of Digital CIM Architectures," in *Proceedings of Design Automation Conference (DAC)*, 2025.

[17] Y. Li, J. Wang, D. Zhu, J. Li, A. Du, X. Wang, Y. Zhang, and W. Zhao, "APIM: An Antiferromagnetic MRAM-Based Processing-In-Memory System for Efficient Bit-Level Operations of Quantized Convolutional Neural Networks," *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 43, no. 8, pp. 2405–2410, 2024.

[18] F. Tu, Y. Wang, Z. Wu, W. Wu, L. Liu, Y. Hu, S. Wei, and S. Yin, "TensorCIM: A 28nm 3.7 nJ/Gather and 8.3 TFLOPS/W FP32 Digital-CIM Tensor Processor for MCM-CIM-Based Beyond-NN Acceleration," in *Proceedings of International Solid-State Circuits Conference (ISSCC)*, 2023.

[19] X. Fu, Q. Ren, H. Wu, F. Xiang, Q. Luo, J. Yue, Y. Chen, and F. Zhang, "P^{3}ViT: A CIM-Based High-Utilization Architecture With Dynamic Pruning and Two-Way Ping-Pong Macro for Vision Transformer," *Transactions on Circuits and Systems I: Regular Papers (TCAS-I)*, vol. 70, no. 12, pp. 4938–4948, 2023.

[20] Y. He, H. Mao, C. Giannoula, M. Sadrosadati, J. Gómez-Luna, H. Li, X. Li, Y. Wang, and O. Mutlu, "PAPI: Exploiting Dynamic Parallelism in Large Language Model Decoding with a Processing-In-Memory-Enabled Computing System," in *Pro-*

ceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2025.

[21] C. Li, Z. Zhou, S. Zheng, J. Zhang, Y. Liang, and G. Sun, "SpecPIM: Accelerating Speculative Inference on PIM-Enabled System via Architecture-Dataflow Co-Exploration," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[22] S.-S. Park, K. Kim, J. So, J. Jung, J. Lee, K. Woo, N. Kim, Y. Lee, H. Kim, Y. Kwon *et al.*, "An LPDDR-based CXL-PNM Platform for TCO-efficient Inference of Transformer-based Large Language Models," in *Proceeding of International Symposium on High-Performance Computer Architecture (HPCA)*, 2024.

[23] J. H. Kim, Y. Ro, J. So, S. Lee, S.-h. Kang, Y. Cho, H. Kim, B. Kim, K. Kim, S. Park *et al.*, "Samsung PIM/PNM for Transfmer Based AI: Energy Efficiency on PIM/PNM Cluster," in *Proceedings of Hot Chips Symposium (HCS)*, 2023.

[24] H. Lee, M. Kim, D. Min, J. Kim, J. Back, H. Yoo, J.-H. Lee, and J. Kim, "3D-FPIM: An Extreme Energy-Efficient DNN Acceleration System Using 3D NAND Flash-Based In-Situ PIM Unit," in *Proceedings of International Symposium on Microarchitecture (MICRO)*, 2022.

[25] M. Kang, H. Kim, H. Shin, J. Sim, K. Kim, and L.-S. Kim, "S-FLASH: A NAND Flash-Based Deep Neural Network Accelerator Exploiting Bit-Level Sparsity," *Transactions on Computers (TC)*, vol. 71, no. 6, pp. 1291–1304, 2021.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All You Need," in *Proceedings of Advances in neural information processing systems (NeurIPS)*, 2017.

[27] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters," in *Proceedings of SIGKDD international conference on knowledge discovery & data mining (KDD)*, 2020.

[28] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley *et al.*, "DeepSpeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2022.

[29] Z. Yuan, Y. Shang, Y. Zhou, Z. Dong, Z. Zhou, C. Xue, B. Wu, Z. Li, Q. Gu, Y. J. Lee *et al.*, "LLM Inference Unveiled: Survey and Roofline Model Insights," *arXiv preprint arXiv:2402.16363*, 2024.

[30] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, "NeuPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[31] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, "AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[32] M. Zhou, W. Xu, J. Kang, and T. Rosing, "TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer," in *Proceedings of International Symposium on High-Performance Computer Architecture (HPCA)*, 2022.

[33] M. Seo, X. T. Nguyen, S. J. Hwang, Y. Kwon, G. Kim, C. Park, I. Kim, J. Park, J. Kim, W. Shin *et al.*, "IANUS: Integrated Accelerator based on NPU-PIM Unified Memory System," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[34] Z. Yu, S. Liang, T. Ma, Y. Cai, Z. Nan, D. Huang, X. Song, Y. Hao, J. Zhang, T. Zhi *et al.*, "Cambricon-LLM: A Chiplet-Based Hybrid Architecture for On-Device Lnference of 70B LLM," in *Proceedings of International Symposium on Microarchitecture (MICRO)*, 2024.

[35] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. Vijaykumar, "Newton: A DRAM-maker's Accelerator-in-Memory (AiM) Architecture for Machine Learning," in *Proceedings of International Symposium on Microarchitecture (MICRO)*, 2020.

[36] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "OPT: Open Pre-trained Transformer Language Models," *arXiv preprint arXiv:2205.01068*, 2022.

[37] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Hugging-Face's Transformers: State-of-the-art Natural Language Processing," *arXiv preprint arXiv:1910.03771*, 2019.

[38] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *Computer Architecture Letters (CAL)*, vol. 19, no. 2, pp. 106–109, 2020.

[39] "High Bandwidth Memory DRAM (HBM3)," *JEDEC*, 2022.

Cenlin Duan received the B.S. degree in Electronic Science and Technology from University of Electronic Science and Technology of China, Chengdu, China, in 2015, and the M.S. degree in Software Engineering from Xidian University, Xi'an, China, in 2018. She is currently pursuing the Ph.D. degree at the School of Integrated Circuit Science and Engineering, Beihang University, Beijing, China. Her current research interests include processing-in-memory architectures and deep learning accelerators.

Jianlei Yang (Senior Member, IEEE) received the B.S. degree in microelectronics from Xidian University, Xi'an, China, in 2009, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2014.

He is currently a Professor in Beihang University, Beijing, China, with the School of Computer Science and Engineering. From 2014 to 2016, he was a postdoctoral researcher with the Department of ECE, University of Pittsburgh, Pennsylvania, USA. His current research interests include emerging computer architectures, hardware-software co-design, and machine learning systems.

Dr. Yang was the recipient of the First/Second place on ACM TAU Power Grid Simulation Contest in 2011 and 2012. He was a recipient of IEEE ICCD Best Paper Award in 2013, ACM GLSVLSI Best Paper Nomination in 2015, IEEE ICESS Best Paper Award in 2017, ACM SIGKDD Best Student Paper Award in 2020.

Rubing Yang received the B.S. degree in computer science and technology from Beihang University, Beijing, China, in 2023. She is currently pursuing the M.S. degree at the School of Computer Science and Engineering, Beihang University, China. Her research interests include computing-in-memory architectures and large language model acceleration.

Yikun Wang received the B.S. degree in computer science and technology from Beihang University, Beijing, China, in 2022, and the M.S. degree at the School of Computer Science and Engineering, Beihang University, China, in 2025. His current research interests include computing-in-memory architectures and deep learning accelerators.

Yiou Wang received the B.S. degree in computer science and technology from Beijing University of Technology, Beijing, China, in 2022, and the M.S. degree at the School of Computer Science and Engineering, Beihang University, China, in 2025. His current research interests include deep learning compilers.

Lingkun Long received the B.S. degree in Computer Science and Technology from Beihang University, Beijing, China, in 2024. He is currently working toward the M.S. degree in Computer Science and Technology in Beihang University, Beijing, China. His current research interests include Efficient AI and ML systems.

Yingjie Qi received the B.S. degree in computer science and technology from Beihang University, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree at the School of Computer Science and Engineering, Beihang University, China. His research interests include graph neural networks acceleration, compute-in-memory architectures and deep learning compilers.

Xiaolin He received the B.S. degree in software engineering from Beihang University, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree at the School of Computer Science and Engineering, Beihang University, China. His research interests include in-memory computing architectures and compiler optimization techniques.

Ao Zhou received the B.S. and M.S. degree in software engineering from Beijing University of Technology, Beijing, China, in 2018 and 2021, respectively. He is currently pursuing the Ph.D. degree at the School of Computer Science and Engineering, Beihang University, China. His research interests include GNN acceleration, computer architecture, FPGA accelerator, and heterogeneous computing. He is one of the contributors to the popular graph neural network framework PyTorch Geometric (PyG).

Xueyan Wang (Member, IEEE) received the B.S. degree in computer science and technology from Shandong University, Jinan, China, in 2013, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2018. From 2015 to 2016, she worked as a Research Scholar at the University of Maryland, College Park, MD, USA. She is currently an Associate Professor with the School of Integrated Circuit Science and Engineering, Beihang University. She has authored more than 40 technical papers in leading journals and conferences. Her primary research interests are in the area of emerging energy-efficient computing architectures, artificial intelligence chips, and intelligent system. Dr.Wang is a recipient of Young Elite Scientists Sponsorship Program by China Association for Science and Technology.

Weisheng Zhao (Fellow, IEEE) received the Ph.D. degree in physics from the University of Paris Sud, Paris, France, in 2007.

He is currently a Professor with the School of Integrated Circuit Science and Engineering, Beihang University, Beijing, China. In 2009, he joined the French National Research Center, Paris, as a Tenured Research Scientist. Since 2014, he has been a Distinguished Professor with Beihang University. He has published more than 300 scientific articles in leading journals and conferences, such as *Nature Electronics*, *Nature Communications*, *Advanced Materials*, IEEE Transactions, ISCA, and DAC. His current research interests include the hybrid integration of nanodevices with CMOS circuit and new nonvolatile memory (40-nm technology node and below) like MRAM circuit and architecture design.

Prof. Zhao was the Editor-in-Chief for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEM I: REGULAR PAPER from 2020 to 2023.